ParGAL: A Scalable Grid-Aware Analysis Library for Ultra Large Datasets

Robert L. Jacob* Xiabing Xu Jayesh Krishna Tim Tautges Robert Latham
Argonne National Laboratory
Kara Peterson Pavel Bochev
Sandia National Laboratory
Mary Haley
National Center for Atmospheric Research

ABSTRACT

Many fields that employ computation require extensive analysis of the output from a petascale simulation of a grid- (or mesh)-based application in order to complete their scientific goals or produce a visual image or animation. Often this analysis cannot be done in-situ because it requires calculating time-series statistics from state sampled over the entire length of the run or analyzing the relationship between similar time series from previous simulations or observations. The programs that perform this analysis are not nearly as flexible in their choice of grids or as high-performing as the primary applications. We will describe a new Parallel Gridded Analysis Library (ParGAL) that performs data-parallel versions of several common analysis algorithms on data from a structured or unstructured grid simulation. The library builds on several scalable systems starting with the Mesh Oriented DataBase (MOAB). MOAB is a library for representing mesh data that supports structured, unstructured finite element and polyhedral grids and also supports parallel operations on those grids including loading to and from disk using parallel I/O. We are using the Parallel-NetCDF (PNetCDF) library to perform parallel I/O operations between the popular NetCDF format and ParGAL. Finally, we also make use of Intrepid, an extensible library for computing operators (such as gradient, curl, divergence, etc.) acting on discretized fields. The design and performance of ParGAL will be described and an example of its application to climate compared to a widely used tool is given.

Index Terms: Petascale Techniques, Scalability Issues, Data Transformation and Representation

1 INTRODUCTION

Today's petascale systems, such as those operated by the DOE Leadership Computing Facilities at Argonne (ALCF) and Oak Ridge National Laboratories (OLCF), will lead to advances in several scientific fields. Many fields such as climate, nuclear engineering, combustion and plasma physics are using these systems to run large, high-resolution versions of their grid- or mesh-based numerical models [7]. In most applications of those models, new knowledge is only gained after significant analysis is done on the output of the petascale simulation, often referred to as "post-processing". In the case of climate modeling the direct output, which may measure in the terabytes, from a single multi-million core-hour simulation tells little about the climate. It is only after multivariate time-series analysis (post-processing) is performed on that data and compared with other runs and observations that something new can be learned.

The programs currently used to perform this analysis in climate and other fields are often not nearly as flexible or high-performing as the primary applications. They are often single threaded and/or 32-bit and may assume structured grids are being used. They either break or require workarounds for the ultra-large unstructured-grid data that is becoming the norm in computational science. In climate science, they are already a bottleneck [25]. Ultra-large data sets present not just a complexity and performance challenge to current tools, but also a memory challenge. The single threaded programs will typically assume they can read all the data in to memory. This requires further workarounds where the researcher uses command line tools to reduce the size of the data to something that can be held in the memory of single node.

The hardware to scalably analyze multi-terabyte gridded output data is available. Both the ALCF and OLCF have dedicated "Data Analysis and Visualization" (DAV) clusters attached to the same disk as the primary compute platform and containing hundreds of "fat" nodes with powerful CPU's, parallel file systems and large amounts of memory (*lens* at OLCF and *eureka* at ALCF). But there is a distinct lack of analysis software that can take advantage of those platforms. The President's Council of Advisors on Science and Technology (PCAST) 2010 review of Networking Information Technology Research and Development (NITRD) [12] said that one of the major challenges in data analysis was "computational models and languages suited for expressing data analysis algorithms that map onto large-scale, parallel systems."

Programs such as Parallel-R [18] provide data-parallel versions of some of its statistical analysis functions. However it does not support operations on a grid. To accurately calculate gradients and other features from output data, it is necessary for the tool to have a representation of the discretization used in the original model. Tools such as GLEAN [24] or DIY [16] provide facilities for data staging and movement in an HPC environment but not the grid-aware data model we need.

In this paper, we describe a new Parallel Gridded Analysis Library (ParGAL,§3) that is built on a computational model that can map onto large analysis clusters (or petascale systems) and explicitly represent the discretizations used in the models. ParGAL provides high-level parallel algorithms that can operate on structured or unstructured grid data in parallel. The library builds on several existing scalable systems (§2) for its data model, algorithm expression and I/O. We are using ParGAL to build a data-parallel version of a popular domain-specific analysis and visualization scripting language which will both allow it to scale and operate on multiple grid types. (§4).

2 COMPONENTS OF PARGAL

Many of the features we wanted ParGAL to have, such as a data model for structured and unstructured grids and a way to define operations within and across grids, were already implemented in other systems. Although all the ParGAL code is new, it has been built on several existing pieces of software.

2.1 MOAB

MOAB is a library for query and modification of structured and unstructured mesh, and field data associated with the mesh[23]. MOAB can represent all entities typically found in the finite element zoo, as well as polygons and polyhedra. Structured mesh is supported as well, with a special interface providing parametric

*e-mail: jacob@mcs.anl.gov

block information[21]. The data model implemented by MOAB references four distinct data types:

- **Entity:** vertices, triangles, quads, etc.
- **Entity Set:** arbitrary collection of entities and other sets
- **Interface:** object through which all other functions are called, i.e. the database
- **Tag:** information stored on Entity, Entity Set, and Interface objects

This data model has proven remarkably versatile, able to represent most semantic information associated with typical meshes, including boundary conditions, solution fields, geometric associativity, and parallel partitions. Internally, MOAB uses an array-based storage model; this allows efficient access to and iteration over fields associated with mesh entities, including vertex- and element-based variables. MOAB uses the HDF5 library for its native save/restore format[1].

For parallel access, mesh is represented and queried in MOAB as a serial mesh local to a processor, with information about the parallel nature of the model accessed (and stored) in the form of sets and tags. For convenience, MOABs `ParallelComm` class also has functions that provide this data, and for performing commonly needed parallel functions. For any entity shared with other processor(s), MOAB stores both the remote processor rank(s) and the handle(s) of the entity on those processor(s), on all processors sharing the entity[22]. Mesh models are initialized in parallel by reading mesh from a single file in parallel, using a partition stored as entity sets in the file. A partitioning tool has been implemented by interfacing with the Zoltan partitioning library[8].

The underlying structured grid representation in MOAB stores connectivity information implicitly, for memory efficiency, while storing vertex locations explicitly, for generality. For ParGAL, specific enhancements were made to MOAB in the area of structured grid representation and efficiency:

- A new convenience API was added to MOAB for accessing structured grid parametric information directly[20]; in addition, structured grid "boxes"[1], and parametric information about each box, is stored in the form of entity sets and tags on those sets, so that it is accessible through the standard MOAB data model.
- Various strategies were implemented for partitioning a structured mesh over processors; these strategies have different characteristics in terms of load balance, size of communication interface between processors, and how closely the layout of vertex or element fields in memory matches the layout of those variables when stored on disk.
- The process of finding shared mesh interfaces in the parallel representation of a mesh was optimized to take advantage of structured grid information. Given the partition method and parametric bounds of sub-domains, handles for mesh vertices on sub-domain boundaries can be computed directly and matched between neighboring processors. In many cases this reduces the time required to initialize a parallel structured mesh by more than an order of magnitude[22].

[1]A structured grid box is a rectangular region of structured grid accessible through an *i, j, k* parameterization.

2.2 Intrepid

Intrepid is a Trilinos [11] package for advanced discretizations of Partial Differential Equations (PDEs) [3]. The abstract framework for compatible discretizations [4] provides the mathematical foundation of Intrepid. This framework prompted reevaluation of conventional software design for PDEs, which usually focuses on a single discretization paradigm. In contrast, Intrepid aims to translate mathematical similarities between finite elements, finite volume and finite difference methods, identified in [4] into software-based similarities. Intrepid has been used to implement numerical methods for PDEs ranging from mimetic least squares for magnetostatics [6] to control volume finite element methods for semiconductor equations [5].

Intrepid offers a wide range of cell-based tools for the implementation of finite element, finite volume and finite difference methods for PDEs. The package represents a *middleware* between higher-level software infrastructures and lower-level cell-based numerics for, e.g., evaluation of basis functions, coordinate transformations, surface parameterizations, and integration of fields over cells, cell faces and cell edges. Intrepid is designed to operate locally on batches of cells having the same topology and data type. A key aspect of the design is that Intrepid separates cell topology from the reconstruction, i.e., the field evaluation process. In other words, a reconstruction "basis" and its evaluation points are not tied to a particular cell topology. This design approach allows Intrepid users to "mix and match" cell topologies with reconstruction operators ("bases") and evaluation points, which enables a virtually unlimited generation of new discretization methods from a small number of basic components.

The ability to "mix and match" an extensive range of fields, cells and evaluation points enables Intrepid to interpret and evaluate virtually any kind of numerical data generated by computer simulations. This makes the package a powerful and flexible tool for data analysis and processing. The ParGAL effort is the first utilization of Intrepid in this application context. ParGAL uses Intrepid to implement forward data operations such as computation of divergence and vorticity from a given velocity field and interpolation between different grids. In addition, ParGAL takes advantage of the discretization capabilities of Intrepid to implement operations such as computation of a stream function and a velocity potential from a given velocity field.

Traditional spherical harmonics approaches for these tasks require global data. In contrast, because Intrepid is rooted in local cell-based operations it does not require global data and can compute the stream function and the potential on any limited domain. When combined with MOAB support for parallel mesh-based communication, the local nature of Intrepid operations makes the combination particularly well-suited for parallel analysis of simulation data.

2.3 PNetCDF

The climate community makes heavy use of the NetCDF self-describing portable file format and its associated programming interface [17]. Portable in this context means the dataset can be moved from machines with different byte-endianess or datatype sizes without needing to change the client code reading or writing those files. Self-describing means the dataset has enough internal structure that client code can use the associated APIs to determine the kind and quantity of variables contained in the dataset. Further, the NetCDF library provides a means for assigning "attributes" (metadata) to variables, dimensions and datasets, offering yet more documentation for the data contained therein. Collaborators at different institutions running on different computing resources rely on both the self-describing and portability features of NetCDF to understand colleagues' work now and in the future.

For parallel I/O needs, the Parallel-NetCDF project [13] provides a parallel programming interface. Parallel-NetCDF maintains the same NetCDF file format and same concepts of attributes, dimensions, and variables, but provides a alternate (though similar) API for parallel programming. This alternate API introduces MPI concepts such as communicators and "info" tuning parameters, but retains the spirit of the serial API. Parallel I/O happens through the MPI-IO [14] interface, but the library can abstract away details such as file views and MPI datatypes. Parallel-NetCDF emphasizes "collective I/O", where all processes participate in an I/O operation. Typically, the MPI-IO library can apply several powerful optimizations to a collective I/O workload. Particularly useful are so-called "deferred mode" parallel operations, where the application specifies a series of data read operations, then frees Parallel-NetCDF to execute them all asynchronously. This allows the library to combine both I/O and communication operations for maximum efficiency.

While Parallel-NetCDF (and serial NetCDF) provide a good interface for regular array access, climate analysis models have grown more sophisticated in the years since these I/O libraries were first designed. Integrating Parallel-NetCDF into MOAB, discussed in Section 2.1, allows us to support these more sophisticated analysis models. MOAB provides the richer description of the grids used in climate analysis, and Parallel-NetCDF provides the optimized parallel I/O for that analysis.

3 ParGAL Architecture

ParGAL leverages the capabilities of the Mesh Oriented datABase (MOAB), Parallel-Netcdf (PNetCDF) and Intrepid libraries, to accomplish efficient, parallel, discretization-accurate data analysis. The design features of the current implementation of ParGAL are:

- **Modularity** Its modular design enables various components to easily interact with each other.
- **Scalability** ParGAL is performance- and scalability-oriented. Each algorithm either implements the best known parallel algorithm or is otherwise carefully designed in order to achieve the highest scalable performance.
- **Portability** The codes strictly adhere to the C++ Standard [19] which is meant to be portable across various parallel systems.
- **Generality** ParGAL is designed to simplify implementation and evaluation of a wide variety of discretization-specific algorithms on a wide range of grid types.
- **Large Scale Data Sets** It has been designed for handling very large scale structured and unstructured grid data sets.

ParGAL is designed to greatly outperform conventional single threaded analysis tools. Its interfaces are designed to encapsulate details about file reading, parallel partitioning, and mesh-based parallel communication, so that the application designer can focus on analysis.

3.1 Software Architecture

There are four main components in ParGAL: Fileinfo, PcVAR, Analysis and Support. They served as the building blocks within ParGAL and also for the user applications. Figure 1 illustrates the architecture of ParGAL and the interaction among each component. The details of each component are given below.

3.1.1 Fileinfo

The Fileinfo class provides an abstraction of a single file or multiple files and a higher level interface to hide lower level details of file management, including opening and closing a file, looking up

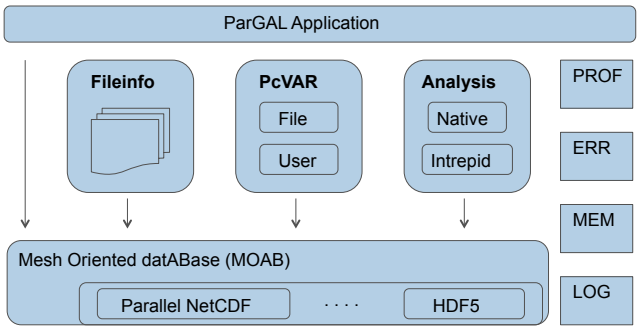

Figure 1: ParGAL Architecture.

which file contains a user-specified time step, retrieving information about file metadata, etc. It also expands the capability of the lower level libraries used. The current MOAB NetCDF/Parallel-NetCDF reader stores file metadata for a single series of files. With Fileinfo, multiple instances of the class can be used to store file metadata for multiple different file series.

3.1.2 PcVAR

ParGAL is designed to work with various large scale structured and unstructured numerical grids. A MOAB mesh instance serves as the database or container for most of the "heavy" data, while ParGAL provides a higher-level index and summary of that data. PcVAR is built on top of MOAB to encapsulate the details of variable data access. For instance, it keeps track of whether a specific time step is loaded. If not, MOAB will be used to load data into the memory and a marker will be set to facilitate later access. Otherwise, the marker will be returned. The distinction between variables from a file and variables created by a user is necessary because the results of some analysis routines need to be stored and the lower level implementation needs to know whether to go to disk for the data or just allocate the space in memory if it is the first time the data is accessed.

Algorithm	Description
max_element	return the maximum element of a variable.
min_element	return the minimum element of a variable.
dim_avg_n	computes the average of a variable's given dimension at all other dimensions.
dim_max_n	computes the maximum of a variable's given dimension at all other dimensions.
dim_min_n	computes the minimum of a variable's given dimension at all other dimensions.
dim_median_n	computes the median of a variable's given dimension at all other dimensions.
vorticity	calculates vorticity from a velocity field on a rectilinear lat/lon grid. Intrepid is used to calculate the partial derivatives assuming a bilinear approximation of velocity on a grid cell.
divergence	calculates divergence from a velocity field on a rectilinear lat/lon grid. Intrepid is used to calculate the partial derivatives assuming a bilinear approximation of velocity on a grid cell.
gather	gather the value of a variable to root 0.

Table 1: ParGAL Function Table

3.1.3 Analysis

The analysis module contains the analysis routines implemented. Most of them will take PcVAR's as input arguments and output results either into a scalar or another PcVAR, similar to how C++ STL's generic algorithm works. The analysis functionality is divided into two categories, native and Intrepid-based. Native algorithms, implemented with functionality provided by ParGAL or MOAB, involve mostly straightforward data-parallel arithmetic operations on mesh-based fields, while Intrepid algorithms are used for more complex discretization-based algorithms. Table 1 shows the algorithms that we have implemented so far and their functionality.

3.1.4 Support Functions

Support Functions include four major modules. ERR is for program errors. We are using C++ exception handling mechanisms and the exception thrown also contains the file name and source line number where the exception is thrown. The LOG module provides logging functionality, the PROF module is for performance profiling and the MEM module is for memory specific operations.

3.2 Code Samples

```
1 double min_element(const pcvar& var) {
2   // compute time steps
3   const std::vector<int>& tsteps = var.get_tsteps();
4   std::vector<double> min_tsteps(tsteps.size());
5
6   // size for each time step
7   std::size_t var_sz = var.get_size();
8
9   // compute min at each time step
10  for (std::size_t i = 0; i != tsteps.size(); ++i) {
11    double* var_ptr = var.get_storage(tsteps[i]);
12    min_tsteps[i] = *std::min_element(var_ptr,
13                                      var_ptr+var_sz);
14    var.delete_storage(tsteps[i]);
15  }
16  double res = 0.0;
17  double loc_min = *std::min_element(min_tsteps.begin(),
18                                     min_tsteps.end());
19
20  // compute global minimum
21  MPI_Reduce(&loc_min, &res, 1, MPI_DOUBLE,
22             MPI_MIN, 0, MPI_COMM_WORLD);
23  return res;
24}
```

Listing 1: Algorithm to find the minimum element.

With the current ParGAL design, the various native and Intrepid-based algorithms can be implemented in a very succinct way. Listing 1 illustrates the implementation of min_element, which returns the minimum element of a given PcVAR variable.

The interface takes a const reference to a PcVAR variable, and returns the minimum element as a double. The time steps associated with the variable can be queried by calling the get_tsteps interface shown on line 3. The vector min_steps on line 4 will be used to store the minimum elements of all time steps for each processor. Line 7 computes the size for each time step. Through a for loop over all the time steps, the minimum element of each time step is stored in the vector min_steps. The variable res will store the global minimum element. Line 17 computes the minimum within each process. Finally a MPI_reduce function (line 21) is used to compute the global minimum element over all processes and the result res is returned (line 23).

4 Application to Climate: ParNCL

To demonstrate the ability of ParGAL to encapsulate parallel analysis at a high level, we are using ParGAL to create a data-parallel version of the NCAR Command Language (NCL). NCL [15] is a free interpreted language that is widely used for data analysis and visualization especially in the climate community. NCL offers a wide array of data analysis operations ranging from simple math operations like finding the minimum element in an array to sophisticated domain-specific operations. The two-dimensional plots rendered by NCL are publication quality and highly customizable (climate scientists use two-dimensional figures instead of three-dimensional visualizations because the aspect ratio of their system is very small). Climate scientists collectively have developed thousands of lines of NCL scripts to perform post processing on the output from climate models and to analyze and visualize climate data. We have developed a parallel version of the NCL interpreter, ParNCL, that performs data analysis in parallel using ParGAL and MOAB.

ParNCL reads the climate data from NetCDF files using MOAB and performs data analysis using the ParGAL library. So far, we have not modified any visualization algorithms in the NCL interpreter. Once the data analysis is complete the single threaded visualization algorithms are used to plot the results.

```
1  MultiDVal dim_min_n(MultiDVal md, int dim)
2  {
3    // Get pcvar corresponding to the multi-dimensional
4    // variable
5    pcvar* var = GetPCVarFromMDVal(md);
6
7    // Get info about the pcvar
8    const std::vector<pcdim>& dims = var->get_dims();
9    const std::vector<int>& tsteps = var->get_tsteps();
10
11   ...
12   // Create a result pcvar to hold the minimum val
13   pcvar *min_var = new pcvar(min_var_name,
14                              dt,
15                              min_var_dims,
16                              ...);
17
18   // Calculate dim_min_n over given dim
19   dim_min_n(*(var), *(min_var), dim);
20
21   ...
22   // Create the result multi-dimensional var
23   MultiDVal res_md = CreateMultiDVal(min_var,
24                                      ...);
25
26   return res_md;
27 }
```

Listing 2: Finding the minimum of a multi-dimensional variable for a given dimension over all other dimensions using ParGAL.

The multi-dimensional variables read from the NetCDF files are stored in a parallel mesh database provided by MOAB. A PcVAR variable is created for each of these multi-dimensional variables and stored with it. This PcVAR variable is used for all data analysis operations that use ParGAL.

Listing 2 illustrates how ParNCL uses ParGAL to find the minimum of a multi-dimensional variable for a given dimension over all other dimensions. In the pseudo code MultiDVal is an internal data structure in ParNCL (and NCL) used to store multi-dimensional variables. To calculate the minimum, first the PcVAR corresponding to the multi-dimensional variable is retrieved as shown in line 5. Then a new PcVAR is created to store the result, which resides on the mesh, as shown in line 13. The minimum is calculated using the data analysis function, dim_min_n() (Table 1), provided by ParGAL. To plot these variables the data corresponding to the variable is gathered using a gather() (Table 1) function, provided by

ParGAL, before passing it to the NCL visualization algorithms.

4.1 Comparison of ParNCL Vorticity Calculation

In this section we compare the performance of ParNCL with NCL on a typical analysis function. As discussed above, ParNCL uses ParGAL and performs data analysis in parallel while NCL performs the data analysis using a single thread. We compare the performance of the single threaded NCL interpreter with ParNCL using the NCL function uv2vrF(), a function that computes the vorticity given the u and v wind components on a fixed rectangular grid.

The native NCL vorticity function uses spherical harmonic analysis [2] which requires global data on a structured spherical grid and provides a very accurate representation of vorticity. However, it is not applicable to data on limited domains or on unstructured grids and is not easily parallelizable. In contrast, the algorithm in ParGAL has been developed using a finite element approach that is highly parallelizable, works equally well on global and limited domains, and is easily extensible to unstructured grids. In the ParGAL approach a formal L^2 projection is used to approximate the vorticity from a nodal velocity field. This method generates a simple linear system whose components are obtained by integrating over cells, thereby eliminating the pole singularity in the case when nodes are located at the poles. The implementation of the algorithm uses Intrepid to provide basis function definitions, numerical quadrature rules, and cell-based numerical operations. The linear system is solved using an iterative solver from the AztecOO package [10] and a multi-level preconditioner from the ML package [9] both part of the Trilinos framework [11].

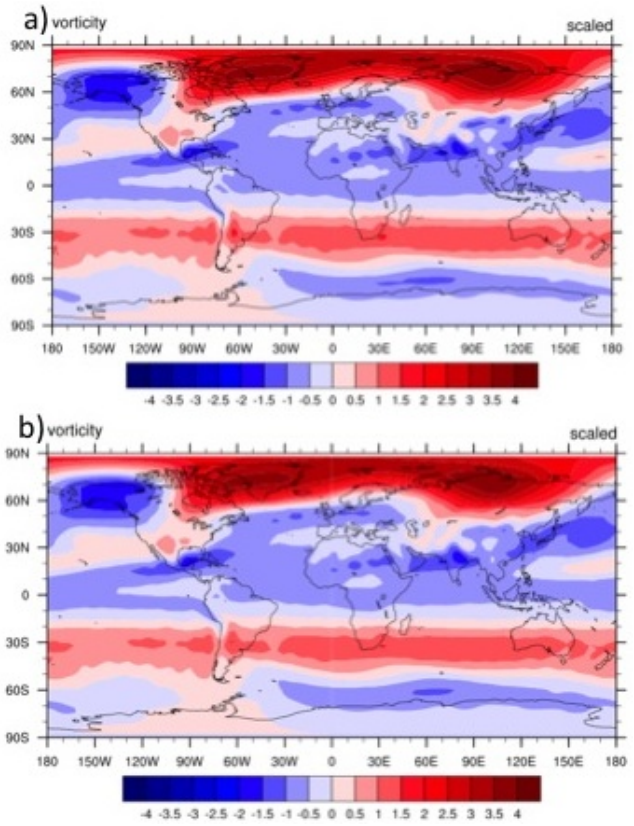

Figure 2: NCL visualization of vorticity calculated from the same U and V field by (a) the original NCL routine and (b) the ParGAL routine

Figure 2 shows that the two algorithms produce nearly identical results visually. To compare performance, we measured the time taken to compute vorticity with both the original spherical harmonic NCL function and the ParGAL function as called by our parallel version of NCL, ParNCL. We used a single time-step data set from an atmospheric general circulation model with a horizontal grid of 384x576 points and 26 vertical levels. The two-dimensional vorticity field is calculated separately on each level. We compared the time for a structured grid because the NCL algorithm can only work on structured grids. Performance results were obtained from

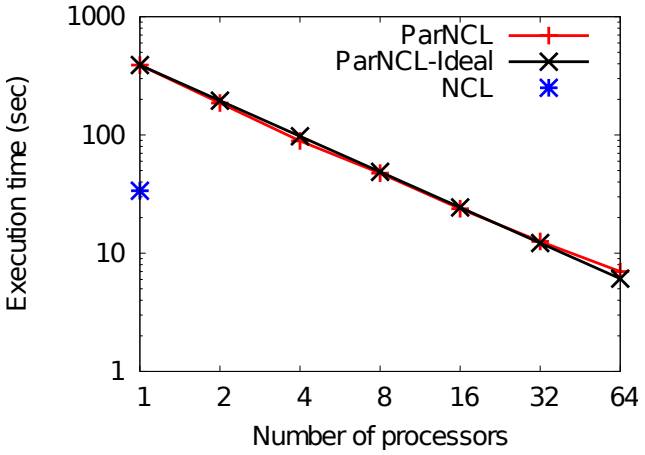

Figure 3: Total Execution time for calculating the vorticity field on each level of 384x576x26 grid vs. number of processors

the *Fusion* cluster at Argonne National Laboratory. Each compute node in the cluster has two Nehalem 2.6GHz Pentium Xeon processors (8 cores), 36GB of memory and uses the Infiniband QDR network for communication. We compared NCL version 6.0.0 Beta with ParNCL (developed from NCL version 6.0.0 Beta). ParGAL was compiled with MVAPICH2 1.4.1, PNetCDF 1.2.0, MOAB 4.5 and Trilinos 10.6.4.

Fig 3 shows that the ParGAL vorticity algorithm scales very well. It does not start out as fast as the NCL algorithm because ParGAL's is more general and there is a cost to that generality. Also at this point we have only focused on correctness and functionality in ParGAL and not performance. However it is still possible to surpass the NCL performance on a modest number of cores. While 64 processors is small in the petascale age, introducing distributed memory algorithms for regular use on any number of processors will be a paradigm shift for most analysis workflows. Also DAV clusters are still shared systems and its unlikely data analysis will be routinely performed on the entire machine.

5 Conclusion

Post-processing analysis of petascale model output is a crucial component of the scientific process in computational science. ParGAL is a library for performing many analysis functions that introduces both the ability to employ data-parallelism and operate on both structured and unstructured grids. To build ParGAL, we have leveraged several well-engineered software libraries that provide key capabilities in the area of parallel mesh and mesh-based data, parallel I/O, and mesh-based discretizations. Using libraries for this purpose not only simplifies construction of an integrated data analysis capability, but makes the post-processing operations similar (both mathematically and algorithmically) to the operations in the original simulation. Our early results comparing performance to a well established visualization and analysis package are encouraging. ParGAL provides high-level functions that hide the details of the distributed memory parallelism from the end user, potentially allowing the familiar script-based analysis approach to scale in par-

allel. NCL has over 300 built-in functions, and our plan is to implement data-parallel versions of the most widely used ones. We will also add the ability to work with additional file formats and grid types. We believe ParGAL and its core set of functions will significantly improve the ability of scientists to gain knowledge from their model simulations.

ACKNOWLEDGEMENTS

This work is part of the Parallel Analysis Tools and New Visualization Techniques for Ultra-Large Climate Data Sets (ParVis) project supported by the Earth System Modeling Program of the Office of Biological and Environmental Research of the U.S. Department of Energy's Office of Science. The project is co-sponsored by the U.S. National Science Foundation via contributions from the National Center for Atmospheric Research, Boulder, CO. We gratefully acknowledge the computing resources provided on "Fusion," a 320-node computing cluster operated by the Laboratory Computing Resource Center at Argonne National Laboratory. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

REFERENCES

[1] Hierarchical data format version 5. http://www.hdfgroup.org/HDF5, Sept. 2011.

[2] J. C. Adams and P. N. Swarztrauber. Spherpack 3.0: A model development facility. *Monthly Weather Review*, 127:1872–1878, 1999.

[3] P. Bochev, H. Edwards, R. Kirby, K. Peterson, and D. Ridzal. Solving pdes with intrepid. *Scientific Programming*, In print., 2012.

[4] P. Bochev and M. Hyman. Principles of mimetic discretizations. In D. N. Arnold, P. Bochev, R. Lehoucq, R. Nicolaides, and M. Shashkov, editors, *Compatible Discretizations, Proceedings of IMA Hot Topics Workshop on Compatible Discretizations*, volume IMA 142, pages 89–120. Springer Verlag, 2006.

[5] P. Bochev and K. Peterson. A new control volume finite element method for the stable and accurate solution of the drift-diffusion equations on general unstructured grids. *Int. J. Num. Meth. Engrg.*, Submitted, 2012.

[6] P. B. Bochev, K. Peterson, and C. M. Siefert. Analysis and computation of compatible least-squares methods for div-curl equations. *SIAM Journal on Numerical Analysis*, 49(1):159–181, 2011.

[7] J. M. Dennis, M. Vertenstein, P. H. Worley, A. A. Mirin, A. P. Craig, R. Jacob, and S. Mickelson. Computational performance of the ultra-high resolution capability in the community earth system model. *Int. J. High Perf. Comp. Appl.*, 26(5):5–16, 2012.

[8] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, 4(2):9097, 2002.

[9] M. Gee, C. Siefert, J. Hu, R. Tuminaro, and M. Sala. ML 5.0 smoothed aggregation user's guide. Technical Report SAND2006-2649, Sandia National Laboratories, 2006.

[10] M. A. Heroux. AztecOO user guide. Technical Report SAND2004-3796, Sandia National Laboratories, 2007.

[11] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, 2005.

[12] J. P. Holdren, E. Lander, and H. Varmus. Report to the President and Congress: Designing a Digital Future: Federally funded research and development in networking and information technology. `http://www.nitrd.gov/pcast-2010/report/nitrd-program/pcast-nitrd-report-2010.pdf`, 2010.

[13] J. Li, W. keng Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale. Parallel netCDF: A high-performance scientific I/O interface. In *Proceedings of SC2003: High Performance Networking and Computing*, SC '03, pages 39–, Phoenix, AZ, November 2003. IEEE Computer Society Press.

[14] MPI-2: Extensions to the message-passing interface. The MPI Forum, July 1997.

[15] NCL – NCAR Command Language (version 6.0.0) [software]. Boulder, Colorado: UCAR/NCAR/CISL/VETS. `http://dx.doi.org/10.5065/D6WD3XH5`, 2012.

[16] T. Peterka, R. Ross, A. Gyulassy, V. Pascucci, W. Kendall, H.-W. Shen, T.-Y. Lee, and A. Chaudhuri. Scalable parallel building blocks for custom data analysis. In *2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 105–112, 2011.

[17] R. Rew and G. Davis. Netcdf: an interface for scientific data access. *Computer Graphics and Applications, IEEE*, 10(4):76 –82, july 1990.

[18] N. Samatova, M. Branstetter, A. Ganguly, R. Hettich, S. Khan, G. Kora, J. Li, X. Ma, C. Pan, A. Shoshani, and S. Yoginath. High performance statistical computing with Parallel R: Applications to biology and climate modeling. In *Journal of Physics: Conference Series SciDAC 2006*, volume 46, pages 505–509, 2006.

[19] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 2000.

[20] T. J. Tautges. MOAB structured mesh interface. `http://svn.mcs.anl.gov/repos/ITAPS/MOAB/trunk/src/moab/ScdInterface.hpp`.

[21] T. J. Tautges. MOAB wiki. `http://trac.mcs.anl.gov/projects/ITAPS/wiki/MOAB`.

[22] T. J. Tautges, J. Kraftcheck, N. Bertram, V. Sachdeva, and J. Materlein. Mesh interface resolution and ghost exchange in a parallel mesh representation. In *Workshop on Large-Scale Parallel Processing, held at the IEEE International Parallel and Distributed Processing Symposium*, Shanghai, China, May 2012. IEEE.

[23] T. J. Tautges, R. Meyers, K. Merkley, C. Stimpson, and C. Ernst. MOAB: a Mesh-Oriented database. SAND2004-1592, Sandia National Laboratories, Apr. 2004. Report.

[24] V. Vishwanath, M. Hereld, and M. E. Papka. Toward simulation-time data analysis and I/O acceleration on leadership-class systems. In *2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 9–14, 2011.

[25] W. Washington. Scientific grand challenges: Challenges in climate change science and the role of computing and the extreme scale. `http://science.energy.gov/~/media/ber/pdf/Climate_report.pdf`, 2008.